A FUNCTIONALLY DISTRIBUTED-PROGRAM KERNEL FOR EMBEDDED REAL-TIME MULTI-PROCESSOR SYSTEMS

J E Cooling and S S Al-Khayatt
Department of Electronic and Electrical Engineering

SUMMARY

Real-time embedded, multi-processor systems are one application area where response times, throughput, reliability and fault tolerance constitute the major design criteria. To meet these needs the overall function (job) can be partitioned into cooperating sub-tasks which are executed concurrently and asynchronously on different processors. Unfortunately, currently available software tools for supporting such designs are primitive, inappropriate or insufficiently robust. One area which suffers especially as a result of this is that of the distributed-program operating system.

A kernel is designed here to manage programs that are distributed among a real-time, multi-processor system. Unlike many scientific and commercial applications, it is not intended to support fragmented programs. Instead, the basis for the design is functional partitioning.

The system is a loosely-coupled one (Fig.1 and Fig.2). It is composed of a number of stations (nodes) linked together through a communication bus. Each node consists of two sections:

- The main-processor section (holding the application programs and the kernel).
- The communications section (providing a transparent interface between the main processors of the system).

KERNEL DESIGN FOR A MULTI-PROCESSOR SYSTEM

The kernel is designed to manage and coordinate processes and resources within each node. In addition it handles inter-processor communication through the use of message-passing primitives. The communication section in each node is transparent during node transaction. Its purpose is to interface the node to the system using both hardware and software resources. A fast, robust communication protocol is implemented in the software of the communication processor.

The following items describe the main features of the real-time kernel as designed for this multi-processor system:

- Program partitioning.
- Process scheduling.
- Real-time interrupts.
- Time synchronization across the system.
- Inter-processor communication and synchronisation.

(a) Program Partitioning.

A task is partitioned functionally into a number of processes which run asynchronously and concurrently within the multi-processor system. Each one of these processes forms the 'main' process in each node. Alongside the main process resides a number of 'server' processes for the handling of messages and calls of client 'main' processes residing at remote nodes. The main and server processes run in a 'quasi-concurrent' form within each node.

(b) Process-Scheduling.

A pre-emptive policy is adopted here. However, unlike the usual method, it does not employ the system 'tick' for scheduling. Instead, a real-time interrupt is captured and utilised to switch between the main process and one of the server processes at each interrupt request. Having trapped the interrupt, processor control is transferred to a re-scheduler. This selects the appropriate server process for activation; it then generates a process switch.

(c) Real-Time Interrupts.

Interrupt service routines are invoked as required to service a variety of prioritised interrupts within each node. Apart from the real-time interrupt mentioned earlier, a clock interrupt is handled at every processor 'tick' to coordinate time in each node.

(d) Time Synchronisation across the system.

A certain process may need to be activated after some elapsed time with respect to a process in another node. To maintain this sense of 'program-time' across the system, timers are synchronised periodically throughout the whole system. Time updates are handled through the use of software interrupts.

(e) Communication and synchronisation.

Inter-processor communication is implemented using a variety of message-passing primitives. This is an efficient and secure way for distributed tasks on different nodes to communicate with each other. Communication between the different distributed programs can be synchronous (a process requests some information and waits for a reply), or asynchronous (a process sends a message, then continues). A selection of message-passing constructs are available for such operations (SEND-REQ., SEND-GET, SEND-REPLY, WAIT, and SEND-MES.).

The kernel software has been developed in MODULA-2, using the Logitech compiler. The communication software has also been written in Modula-2, using the FTI compiler. An Intel 80188 processor and a Hitachi 64180 processor are used as the main and communication processors respectively.

BIBLIOGRAPHY

SYSTEM CONFIGURATION

Station A

Main Processor

Station B

Main Processor

Station C

Main Processor

Fig. 1

SYSTEM CONFIGURATION

Station Configuration

Main Processor

Random Access Memory

Main Processor

Random Access Memory

Random Access Memory

Fig. 2

SYSTEM CONFIGURATION